Large-scale and user-friendly exact diagonalization in Chapel

Tom Westerhout
tom.westerhout@ru.nl
Institute for Molecules and Materials,
Radboud University
Nijmegen, The Netherlands

Mikhail I. Katsnelson
m.katsnelson@science.ru.nl
Institute for Molecules and Materials,
Radboud University
Nijmegen, The Netherlands

Bradford L. Chamberlain
bradford.chamberlain@hpe.com
Hewlett Packard Enterprise
Seattle, USA

ABSTRACT

Exact diagonalization is one of the oldest and most established numerical methods for simulation of small quantum systems. Exponential scaling of the computational resources is the main limiting factor in its applicability, and requires highly parallel implementations if one wants to consider slightly larger systems. In this talk we will discuss the implementation of a scalable and user-friendly exact diagonalization package where Chapel is used for both shared- and distributed-memory parallelism. We will talk about the architecture of our package and progress in its implementation, focusing on multi-language interoperability and usage of Singularity containers for deployment.

KEYWORDS

distributed computing, Chapel, quantum physics, exact diagonalization

1 INTRODUCTION

Quantum mechanics has long been accepted as essential for the description of many experimentally-observed phenomena in condensed matter physics. Mathematically, each system is typically described by a Hermitian operator called the *Hamiltonian* of the system [4]. The dimension of this operator can in general be infinite, but there are nonetheless cases when the Hamiltonian is finite-dimensional; in other words, it is a matrix. An example on which we will focus is the study of localized spins or electrons confined to a lattice.

Exact diagonalization (ED) is one of the oldest approaches to tackle such systems numerically [7]. For example, it has been used to determine phase diagrams of various models at both zero [1, 8] and finite [3] temperatures, to study time evolution of a system after an external perturbation [10], to study decoherence phenomena [2] and many more. Since ED is an exact method, it is commonly used to test and benchmark other algorithms such as tensor network methods [11] or variational Monte Carlo algorithms [13].

The main limitation of exact diagonalization is exponential scaling of computational resources with system size. For instance, for localized spin-1/2 particles, the dimension of the Hamiltonian is $O(2^N)$ where N is the number of particles. There are techniques to reduce the dimension by constant, polynomial, or even small exponential factors, but in the end, the scaling will remain exponential.

In the talk, we will start with a short overview of techniques to reduce the dimension of the Hamiltonian as well as available open-source implementations. We will explain the current limitations of these implementations, and how `Chapel` might help in writing a better one. With user-friendliness and scalability as our main goals, we will present the architecture of our solution and discuss progress in its implementation. In particular, we will focus on multi-language interoperability and the usage of Singularity [6] as two unique aspects of our solution. Finally, we will discuss the current challenges and how we hope to overcome them in the future.

2 BACKGROUND

Mathematically, exact diagonalization can be seen as an eigenvalue problem $Hv = Ev$, where a Hermitial matrix $H \in \mathbb{C}^{D\times D}$ is the Hamiltonian of the system, $v \in \mathbb{C}^D$ is called an eigenvector, and $E \in \mathbb{R}$ is the corresponding eigenvalue (called energy) of the system. The complete solution of such a problem requires $O(D^2)$ memory and $O(D^3)$ time. However, 1) one is often interested in just a few eigenvectors of the system, and 2) the Hamiltonian is typically a very sparse matrix with just $O(D \log_2 D)$ nonzero entries. In such cases, sparse representation of H together with iterative solvers can be used to reduce both memory and time requirements to $O(D \log_2 D)$. Memory requirements can be reduced even further if one resorts to a so called *matrix-free* representation of H where a prescription of how to compute matrix elements is stored rather than the matrix elements themselves. This lowers the memory usage to $O(D)$ which comes from the fact that one still needs to store a few (often, fewer than 10) vectors to build a Krylov subspace of H.

For concreteness, let us now focus on spin-1/2 particles (also called qubits). In general, for such a system $D = 2^N$ where N is the number of particles. However, by using symmetries, the Hamiltonian can be brought into a block-diagonal form and each block can be considered separately. For instance, if H is $U(1)$-symmetric, one can work in sectors of well-defined *magnetization*, and the largest block has the dimension $D = \binom{N}{N/2}$. By making use of the space group of the lattice, D can be reduced further by a factor of 100–1000 (for instance, the space group of a 36-site square lattice contains 288 elements, and the space group of a 48-site triangular lattice contains 1152 elements). In Ref. [18] such techniques have been used to diagonalize a system of 50 spin-1/2 particles. $D \approx 3.2 \times 10^{11}$ was used in the paper such that memory requirements were around 15.5TB. A naive approach would result in $D = 2^{50} \approx 10^{15}$ and require 8PB to just store a single vector.

Good implementation of such symmetry-reduced Hamiltonians in matrix-free form can be a non-trivial task, but there are a few open-source projects that succeed in doing so. `QuSpin` [15] is a Python package that supports arbitrary user-defined symmetries and can work with both localized spin-1/2 particles and fermions. Unfortunately, `QuSpin` currently does not support multi-node parallelism. `HΦ` [5] and `SPINPACK` [12] are two notable projects that do support parallel execution. Both projects rely on MPI and have similar limitations. Firstly, in comparison with `QuSpin`, they are both not very user-friendly. For example, `HΦ` requires the creation of

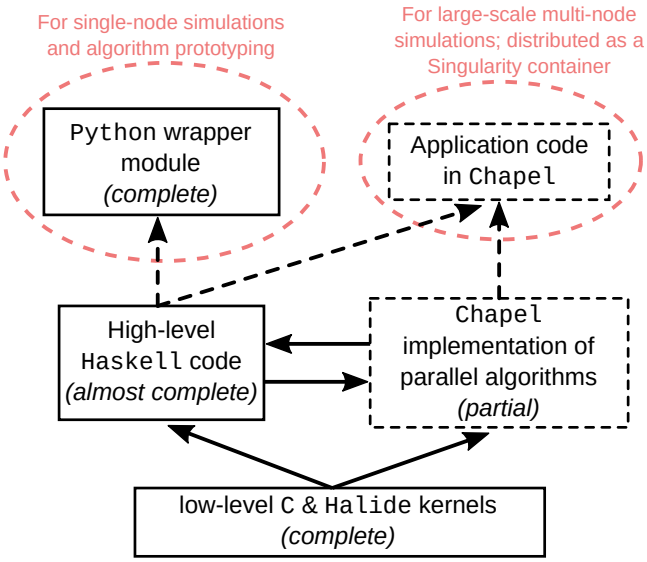

Figure 1: General architecture and implementation status of our solution. With black solid lines and arrows we indicate components which are already implemented; black dashed lines and arrows indicate components which implementation is still a work in progress. Arrows indicate dependencies between components. For example, "Application code in `Chapel`" depends on both "High-level `Haskell` code" and "`Chapel` implementation of parallel algorithms".

many input files with non-standard formatting. The content of the input files is non-trivial either since the user has to manually transform spin operators into fermionic ones. In SPINPACK, one even has to write low-level C code to generate the input files. Furthermore, both packages are rather big (HΦ contains more than 28 000 lines of low-level C code, and SPINPACK — more than 24 000) and have a non-modular structure which makes it difficult to experiment with new algorithms.

We wish to combine the user-friendliness and extensibility of QuSpin with the performance and scalability of SPINPACK. To this end, we choose to extend `lattice-symmetries` library [17]. `lattice-symmetries` is our earlier contribution to open-source solutions for exact diagonalization, and contains a high-performance implementation of routines for working with symmetry-reduced Hamiltonians (outperforming both QuSpin and SPINPACK for single-node simulations). The package currently does not support multi-node simulations, and we have chosen to use `Chapel` instead of the more traditional `MPI+OpenMP` approach to provide scalability while keeping the code simple and readable.

3 APPROACH

Currently, the `lattice-symmetries` package consists of a core written in `C++` and `C`, and a `Python` wrapper module for easy prototyping of new algorithms. The `SpinED` [16] application, written in `Haskell`, is a thin wrapper around `lattice-symmetries` and `PRIMME` [14] libraries and lets users with no programming experience focus on physics and run exact diagonalization simulations. Recently, some low-level kernels in `lattice-symmetries` have been converted from `C++` to `Halide` [9] to also allow compilation for other processor architectures than `x86-64`. Adding yet another programming language to the mix might seem like a questionable endeavour, but we believe that the benefits that `Chapel` brings, outweigh the extra maintenance costs.

Specifically, the Partitioned Global Address Space (PGAS) paradigm allows one to easily prototype distributed algorithms. After ensuring that the algorithms work as expected, one can start optimizing them for performance, keeping the code readable at all times. One of the key features of `Chapel` is that the same code can be compiled for a laptop with just a few cores and a cluster with thousands of nodes and perform well on both. We exploit this feature by compiling our code twice:

(1) For one node and with `--library` mode enabled. This produces a shared library which can be used from the `Python` wrapper module. `Chapel` thus serves as a replacement for `OpenMP`.
(2) For multiple locales and without enabling the `--library` mode. The produced application serves as a replacement for `SpinED` and should be usable for large-scale simulations on compute clusters with many nodes.

This greatly reduces the amount of code which has to be written compared to the more traditional `MPI+OpenMP` model. Furthermore, `Chapel` has good support for calling external `C` functions, and invoking low-level kernels from `lattice-symmetries` is trivial. Calling into `Haskell`, however, cannot be done directly, so our `Haskell` code exposes a `C` interface which is usable from `Chapel`. The general architecture and languages involved in the implementation are depicted in Figure 1. More details on this multi-language interoperability, such as how the cyclic dependency between `Haskell` and `Chapel` code is resolved, will be given in the talk.

Another important point is the packaging and distribution of the application produced in item (2). `SpinED` is distributed as a statically-linked executable such that one can download one file and directly run it. We wish to provide a similar experience, but `Chapel` applications compiled for multiple locales require the `Chapel` distribution to be installed on the target machine (specifically, auxiliary `GASNet` scripts are required to start the application). To this end, we use Singularity to pre-build the project for multiple `GASNet` conduits. We have successfully built such a container for a "Hello world"-type application and more details about it, such as which launcher we use internally, will be given in the talk.

4 OUTLOOK

We have described the architecture for extending the `lattice-symmetries` library to support multi-node parallelism while preserving its user friendliness. There is still much work to be done to fully implement this solution. Currently, the main obstacle is the performance of our distributed algorithms. This is most likely due to the limited experience of the first author with `Chapel` programming language, and we hope to overcome it by collaborating closer with the `Chapel` development team.

REFERENCES

[1] Sylvain Capponi and Andreas M Läuchli. 2015. Phase diagram of interacting spinless fermions on the honeycomb lattice: A comprehensive exact diagonalization study. *Physical Review B* 92, 8 (2015), 085146.

[2] Hylke Donker, Hans De Raedt, and Mikhail Katsnelson. 2017. Decoherence and pointer states in small antiferromagnets: A benchmark test. *SciPost Physics* 2, 2 (2017), 010.

[3] N Elstner and AP Young. 1994. Spin-1/2 Heisenberg antiferromagnet on the kagome lattice: High-temperature expansion and exact-diagonalization studies. *Physical Review B* 50, 10 (1994), 6871.

[4] David J Griffiths and Darrell F Schroeter. 2018. *Introduction to quantum mechanics*. Cambridge university press.

[5] Mitsuaki Kawamura, Kazuyoshi Yoshimi, Takahiro Misawa, Youhei Yamaji, Synge Todo, and Naoki Kawashima. 2017. Quantum lattice model solver HΦ. *Computer Physics Communications* 217 (2017), 180–192. https://doi.org/10.1016/j.cpc.2017.04.006

[6] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PloS one* 12, 5 (2017), e0177459.

[7] HQ Lin. 1990. Exact diagonalization of quantum-spin models. *Physical Review B* 42, 10 (1990), 6561.

[8] Masao Ogata, MU Luchini, S Sorella, and FF Assaad. 1991. Phase diagram of the one-dimensional t-J model. *Physical review letters* 66, 18 (1991), 2388.

[9] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.

[10] Jani Särkkä and Ari Harju. 2008. Spin dynamics in a double quantum dot: Exact diagonalization study. *Physical Review B* 77, 24 (2008), 245315.

[11] Ulrich Schollwöck. 2005. The density-matrix renormalization group. *Reviews of modern physics* 77, 1 (2005), 259.

[12] Jörg Schulenburg. 2017. SPINPACK. https://www-e.ovgu.de/jschulen/spin/.

[13] Sandro Sorella. 2005. Wave function optimization in the variational Monte Carlo method. *Physical Review B* 71, 24 (2005), 241103.

[14] Andreas Stathopoulos and James R McCombs. 2010. PRIMME: PReconditioned Iterative MultiMethod Eigensolver—methods and software description. *ACM Transactions on Mathematical Software (TOMS)* 37, 2 (2010), 1–30.

[15] Phillip Weinberg and Marin Bukov. 2017. QuSpin: a Python package for dynamics and exact diagonalisation of quantum many body systems part I: spin chains. *SciPost Physics* 2, 1 (2017), 003.

[16] Tom Westerhout. 2020. SpinED. https://github.com/twesterhout/spin-ed.

[17] Tom Westerhout. 2021. 'lattice-symmetries': A package for working with quantum many-body bases. *Journal of Open Source Software* 6, 64 (2021), 3537. https://doi.org/10.21105/joss.03537

[18] Alexander Wietek and Andreas M Läuchli. 2018. Sublattice coding algorithm and distributed memory parallelization for large-scale exact diagonalizations of quantum many-body systems. *Physical Review E* 98, 3 (2018), 033309.